# 函数详解



## 学习目标

- 掌握 TypeScript 中的函数类型标注
- 函数可选参数和参数默认值
- 剩余参数
- 函数中的 `this`
- 函数重载

## 函数的标注

一个函数的标注包含

- 参数
- 返回值

```
function fn(a: string): string {};
let fn: (a: string) => string = function(a) {};

type callback = (a: string): string;
interface ICallBack {
  (a: string): string;
}

let fn: callback = function(a) {};
let fn: ICallBack = function(a) {};
```

## 可选参数和默认参数

### 可选参数

通过参数名后面添加 `?` 来标注该参数是可选的

```
let div = document.querySelector('div');
function css(el: HTMLElement, attr: string, val?: any) {

}
// 设置
div && css( div, 'width', '100px' );
// 获取
div && css( div, 'width' );
```

## 默认参数

我们还可以给参数设置默认值

- 有默认值的参数也是可选的
- 设置了默认值的参数可以根据值自动推导类型

```
function sort(items: Array<number>, order = 'desc') {}
sort([1,2,3]);

// 也可以通过联合类型来限制取值
function sort(items: Array<number>, order:'desc'|'asc' = 'desc') {}
// ok
sort([1,2,3]);
// ok
sort([1,2,3], 'asc');
// error
sort([1,2,3], 'abc');
```

## 剩余参数

剩余参数是一个数组，所以标注的时候一定要注意

```
interface IObj {
    [key:string]: any;
}
function merge(target: IObj, ...others: Array<IObj>) {
    return others.reduce( (prev, currnet) => {
        prev = Object.assign(prev, currnet);
        return prev;
    }, target );
}
let newObj = merge({x: 1}, {y: 2}, {z: 3});
```

# 函数中的 this

无论是 `JavaScript` 还是 `TypeScript` ，函数中的 `this` 都是我们需要关心的，那函数中 `this` 的类型该如何进行标注呢？

- 普通函数
- 箭头函数

## 普通函数

对于普通函数而言，`this` 是会随着调用环境的变化而变化的，所以默认情况下，普通函数中的 `this` 被标注为 `any`，但我们可以在函数的第一个参数位（它不占据实际参数位置）上显式的标注 `this` 的类型

```
interface T {
    a: number;
    fn: (x: number) => void;
}

let obj1:T = {
    a: 1,
    fn(x: number) {
        //any类型
        console.log(this);
    }
}


let obj2:T = {
    a: 1,
    fn(this: T, x: number) {
        //通过第一个参数位标注 this 的类型，它对实际参数不会有影响
        console.log(this);
    }
}
obj2.fn(1);
```

## 箭头函数

箭头函数的 `this` 不能像普通函数那样进行标注，它的 `this` 标注类型取决于它所在的作用域 `this` 的标注类型

```
interface T {
    a: number;
    fn: (x: number) => void;
}

let obj2: T = {
    a: 2,
    fn(this: T) {
        return () => {
            // T
            console.log(this);
        }
    }
}
```

# 函数重载

有的时候，同一个函数会接收不同类型的参数返回不同类型的返回值，我们可以使用函数重载来实现，通过下面的例子来体会一下函数重载

```
function showOrHide(ele: HTMLElement, attr: string, value:
'block'|'none'|number) {
    //
}

let div = document.querySelector('div');

if (div) {
  showOrHide( div, 'display', 'none' );
  showOrHide( div, 'opacity', 1 );
    // error，这里是有问题的，虽然通过联合类型能够处理同时接收不同类型的参数，但是多个参数之
间是一种组合的模式，我们需要的应该是一种对应的关系
  showOrHide( div, 'display', 1 );
}
```

我们来看一下函数重载

```
function showOrHide(ele: HTMLElement, attr: 'display', value: 'block'|'none');
function showOrHide(ele: HTMLElement, attr: 'opacity', value: number);
function showOrHide(ele: HTMLElement, attr: string, value: any) {
  ele.style[attr] = value;
}

let div = document.querySelector('div');

if (div) {
  showOrHide( div, 'display', 'none' );
  showOrHide( div, 'opacity', 1 );
  // 通过函数重载可以设置不同的参数对应关系
  showOrHide( div, 'display', 1 );
}
```

- 重载函数类型只需要定义结构，不需要实体，类似接口

```
interface PlainObject {
    [key: string]: string|number;
}

function css(ele: HTMLElement, attr: PlainObject);
function css(ele: HTMLElement, attr: string, value: string|number);
function css(ele: HTMLElement, attr: any, value?: any) {
    if (typeof attr === 'string' && value) {
        ele.style[attr] = value;
    }
    if (typeof attr === 'object') {
        for (let key in attr) {
            ele.style[attr] = attr[key];
        }
    }
}

let div = document.querySelector('div');
if (div) {
    css(div, 'width', '100px');
    css(div, {
        width: '100px'
```

```
    });

    // error，如果不使用重载，这里就会有问题了
    css(div, 'width');
}
```